

Capi: Cloud Computing API

Bruno Costa and Miguel Matos and António Sousa
{blc,mm}@lsd.di.uminho.pt, als@di.uminho.pt

Universidade do Minho, Braga, Portugal

Abstract. Cloud Computing is an emerging business model to provide access to IT resources in a pay per use fashion. Those resources range from low-level virtual machines, passing by application platforms and ending in ready to use software delivered through the Internet, creating a layered stack of differentiated services. Different cloud vendors tend to specialize in different levels, offering different services, each with its own proprietary API. This clearly led to provider lock-in and hinders the portability of a given application between different providers. The goal of this paper is therefore to provide an abstraction of the different levels of services in order to reduce vendor lock-in and improve portability, two issues that we believe impair the adoption of the Cloud Computing model.

1 Introduction

Cloud Computing's [9] key idea is to provide access to arbitrary resources over the Internet which will be otherwise confined to specialized infrastructures. The access to the Cloud is based on a pay per use model, providing resources that range from the hardware level, to software as a service, passing by an intermediate level where applications could be deployed.

Although there are already multiple providers world-wide [3,1,6] providing services at each level of the Cloud stack, there are no standard interfaces to access those services. This lack of standardization leads to vendor lock-in and decreases portability as applications need to be (re-)written to comply with a given API, made available by the Cloud service provider.

In this paper, we propose a general purpose Cloud API (CAPI), that offers programming interface abstractions to each one of the levels of the Cloud stack. With this general API, customers are able to build their applications against it and then, by recurring to the concrete implementation of the provider - the driver - use the underlying services. Additionally, CAPI considers all the abstraction levels, providing the customer with a common interface to manage all of them, using different Cloud providers, at different abstraction levels, in a transparent fashion.

As can be seen in Figure 1, the different abstraction levels of the Cloud stack can be built using the functionalities of the level below it.

In the lowest abstraction level, Infrastructure as a Service (IaaS), the providers offers virtualized hardware solutions, such as storage and processing capacity

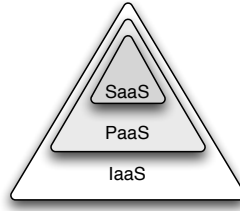


Fig. 1. Cloud services levels.

(e.g. Amazon EC2 [1]). The Platform as a Service (PaaS) level models a software stack - the platform - that could be used to develop, deploy and run applications (e.g. Google App Engine [3] and Force.com [8]). In the less featured level, only a service is provided to the client, which is a remote application available as a web service or as a browser enabled application (e.g. Yahoo! Mail [10] and Google Docs [4]).

The rest of this paper is structured as follows. In Section 2 we present the architecture of the API and the rationale that led to it. Finally Section 3 concludes the paper.

2 Architecture

Taking into account the functional requirements of the system it is clear that we need a modular, pluggable mechanism in which it is possible to control the modules' life-cycle. To fulfill these requirements, our proposal is built atop the OSGi [7] platform. The dynamic properties of OSGi enables partial deployment of modules, as well as changing them at runtime.

In order to enable the seamless integration with management user interfaces, CAPI exports its interfaces through standard Java Management eXtensions (JMX) [5].

Figure 2 depicts the CAPI architecture. It mainly consists in three modules that represent each one of the abstraction levels in the Cloud stack, and an additional module that exposes the functionalities of the other modules through JMX. In addition to these, Monitoring and Security capabilities appear as pervasive to all abstraction levels.

As the API is intended to be general we have to model abstract concepts into the definition of the interface. The rationale behind the proposal is to think in terms of entities, their relationships and the expected behaviour.

2.1 IaaS: Resource Deployment

In the lowest abstraction level we model everything as a *Resource*, either it is a hard disk, a CPU, storage and so on. The basic idea is to manipulate all the

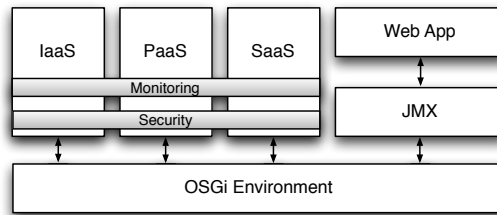


Fig. 2. Architecture Overview.

components available at this level through a common interface with well defined properties.

In this module we define three important interfaces. The *Resource Pack* models image bundles that can be deployed to the provider that includes pre-built images and custom-made images. A *Resource Container* is an abstraction of a running *Resource Pack*, having the ability to add and remove resources at runtime. The *Resource Manager* appears as an entry point to this module, offering capabilities of management and monitoring to the different resources deployed.

In addition to these, resources deployed to the Cloud usually have an internal network encompassing the resources containers of a given customer. This is managed by a *Network* interface, that offers the ability to expose the machines in the internal network to the public network.

2.2 PaaS: Applications Management

This middleware layer support an abstraction for running applications on the Cloud provider platform. In the PaaS you develop applications that can be deployed in the Cloud, following the software development kit offered by the provider. Therefore, we see the applications in the cloud as a set of *Managed Applications* that we are able to manipulate.

The *Managed Applications* have a well defined lifecycle. Thus, it is possible to start and stop that application, as well as observe its state. It also permits to watch each application state and invoke runtime operations exposed by it. To complete the module, the *Platform Manager* controls the bootstrap of all the applications.

2.3 Software as a Service (SaaS): Monitoring Services

The conceptual idea is to provide monitoring capabilities to the users of SaaS. The restriction of functionalities available in this module is directly related to its low flexibility. In this level, the provider offers a complete software solution through the Internet, usually accessible to the customer via a web browser.

All components in SaaS are seen as *Monitored Services* which state we can monitor.

3 Conclusion

In this paper we presented CAPI, a general-purpose Cloud Computing API that covers the different abstraction levels of Cloud services available today. We took an high level approach that focus in the management issues raised when managing components, be it virtual machines or software, in a Cloud environment. With this approach our proposal is able to cope with the offerings of the different providers at each level, and with the imposition of the different abstraction levels on top of the lower ones.

Due to its flexible nature it is possible to build applications that use services from different cloud providers and at different abstraction levels, effectively increasing the portability among different Cloud providers and therefore breaking the lock-in to a specific vendor.

As future work we intend to leverage on the JMX-exposed interfaces to provide a friendly and easy to use web based interface. As the Cloud platform evolves and matures, new abstractions and functionalities will surely arise and thus CAPI must be constantly improved to cope with the new cloud requirements.

References

1. Amazon.com, Inc. Amazon elastic compute cloud. <http://aws.amazon.com/ec2>, 2009.
2. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.
3. Google. App engine. <http://code.google.com/appengine>, 2009.
4. Google. Google Docs. <http://docs.google.com/>, 2009.
5. E. McManus. Java management extensions. Technical report, Java Community Process (JSR'3), 2006.
6. Microsoft Corporation. Azure services platform. <http://www.microsoft.com/azure/default.mspx>, 2008.
7. OSGi Alliance. Osgi service platform. http://osgi.org/osgi.technology/download_specs.asp, 2005.
8. salesforce.com. Force.com. <http://www.salesforce.com/force/>, 2009.
9. L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.
10. Yahoo! Inc. Yahoo! Mail. <http://mail.yahoo.com>, 2009.